Enhancing Security Performance Through IA-64 Architecture

Stephen Moore, Sr. Software Architect
Intel Corporation

Developers' Track
January 17, 2000 3:00PM

Agenda

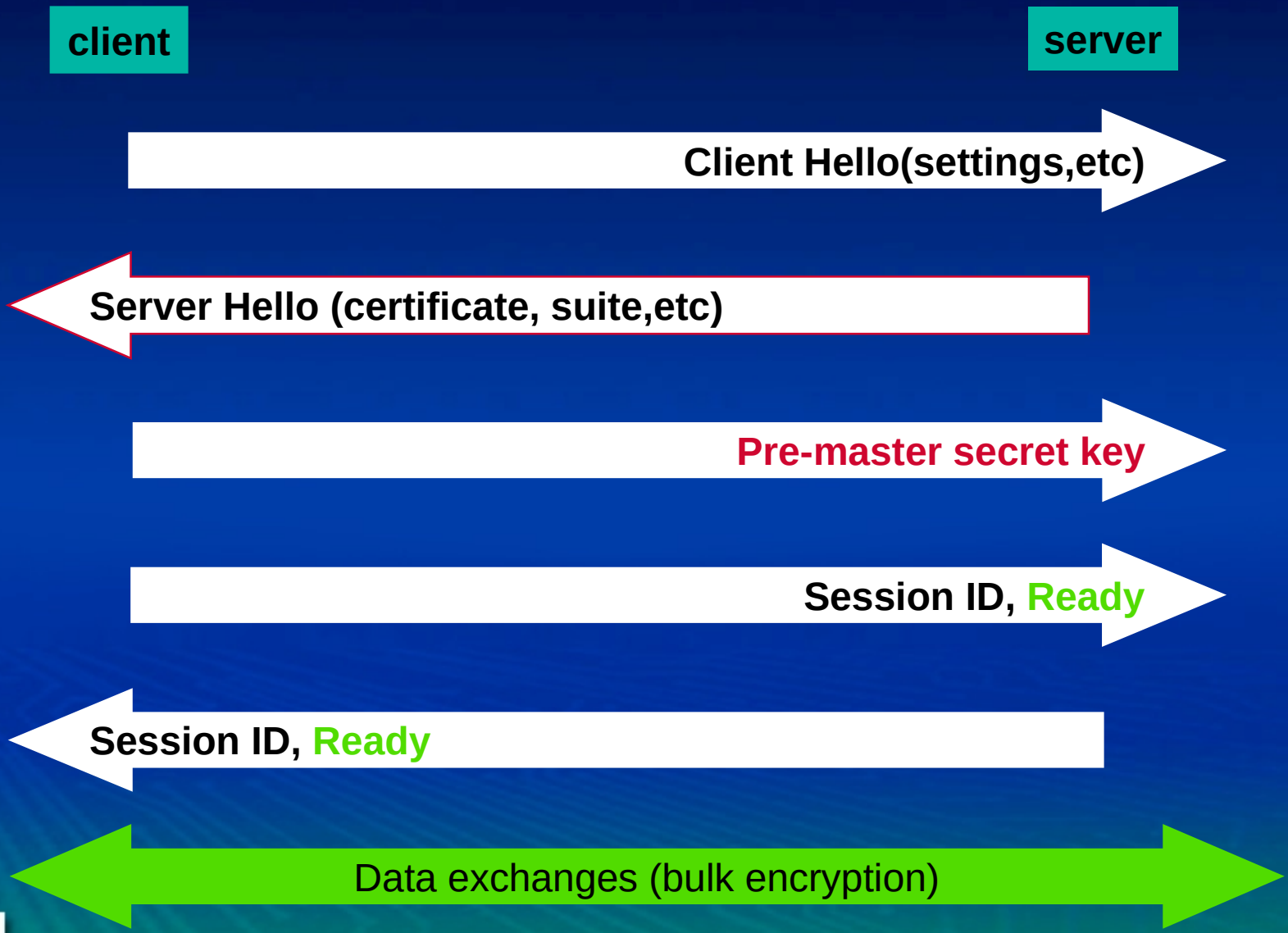
- How security impacts performance for e-Commerce
- Identifying key algorithms for improving performance
- Optimizing on Itanium™ processor

Security impacts performance

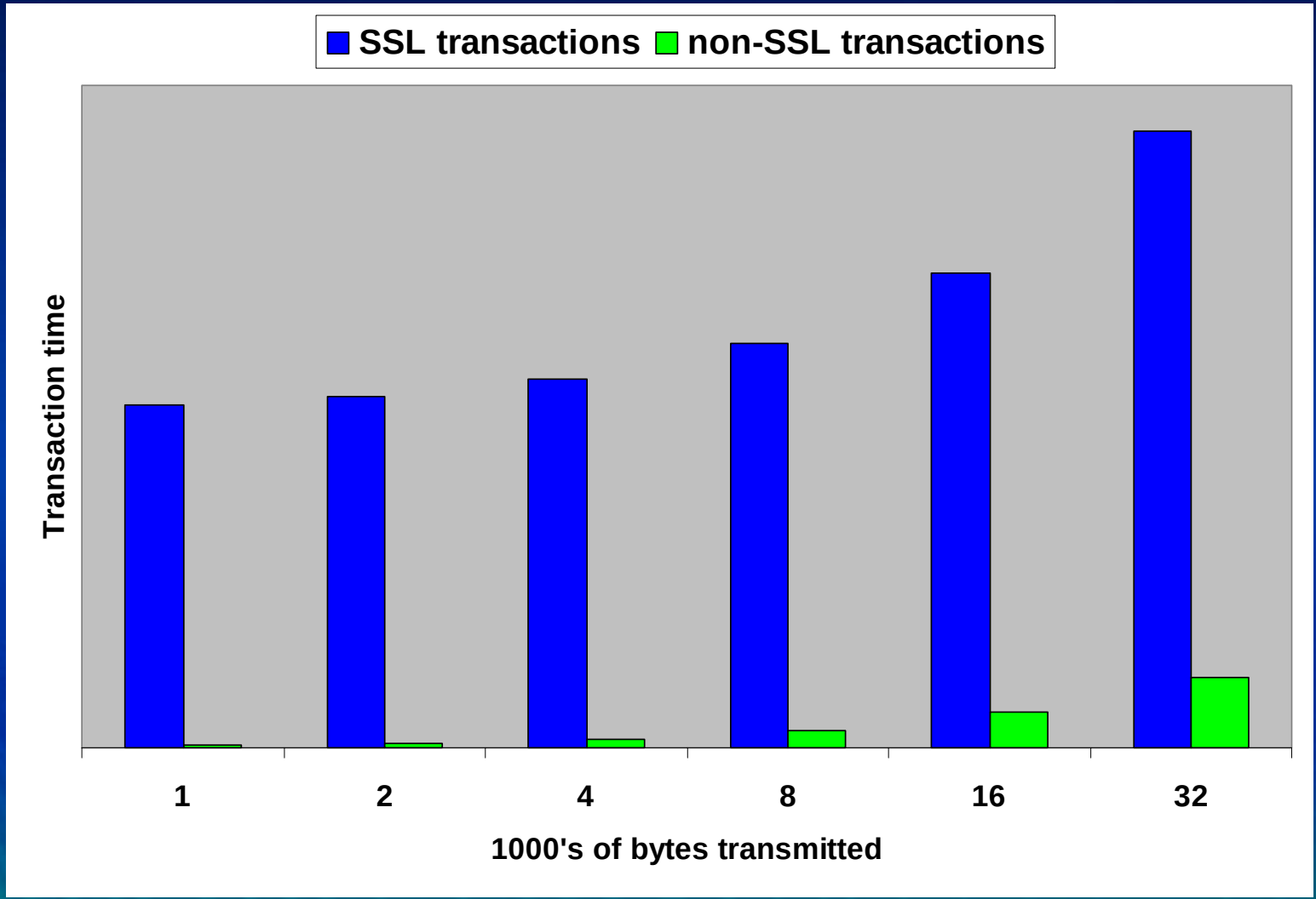
Security and e-Commerce

- Secure transactions enable e-Commerce
- SSL is the standard for secure Web transactions
  - Protocol for secure communication
  - Built upon a core set of algorithms
    - Public-key encryption
    - Message digest
    - Digital signature
    - Bulk encryption

SSL - The basics



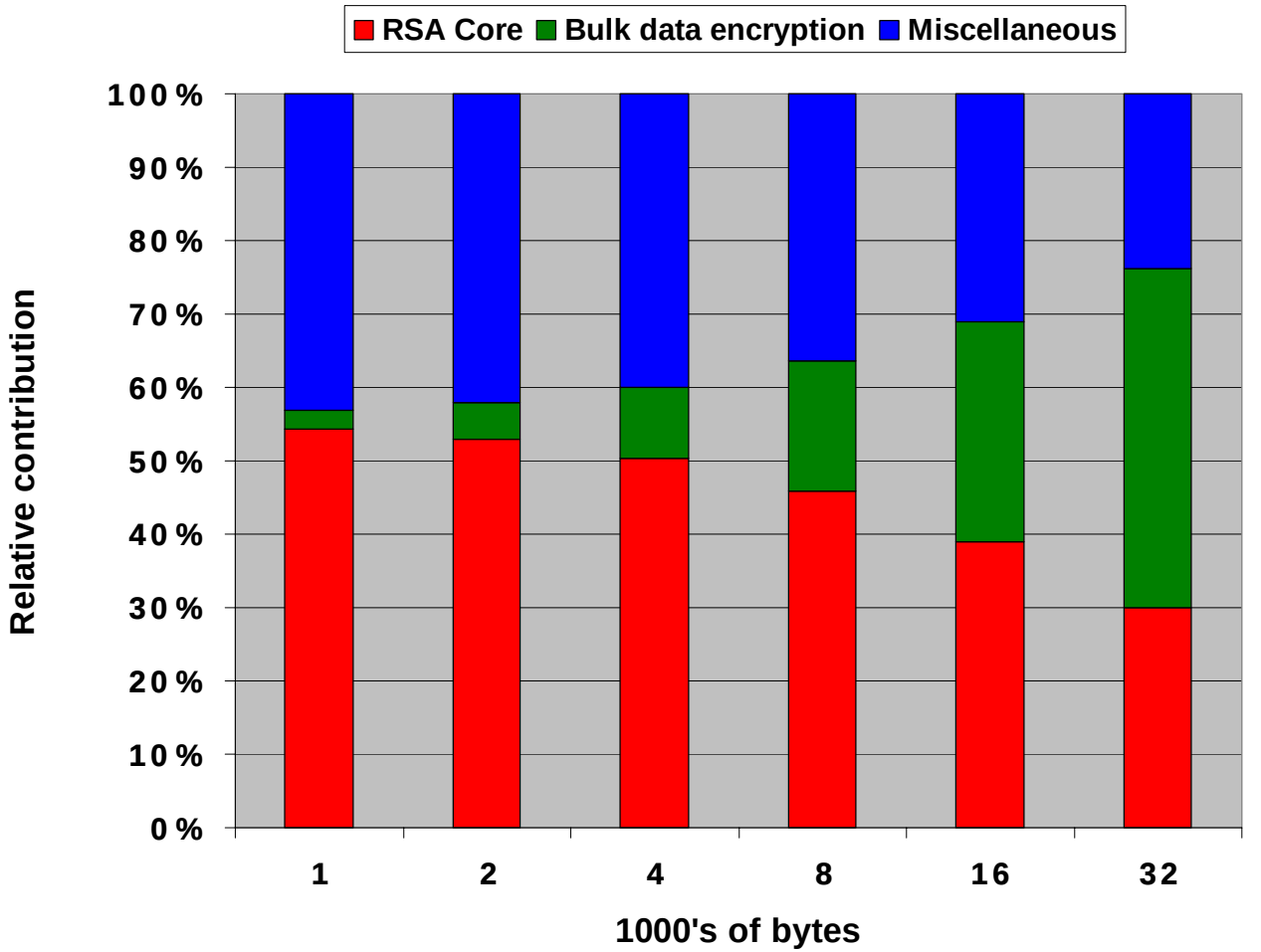
The high cost of SSL



Source - Intel Lab research

Identify Key Algorithms

Where is the time spent?

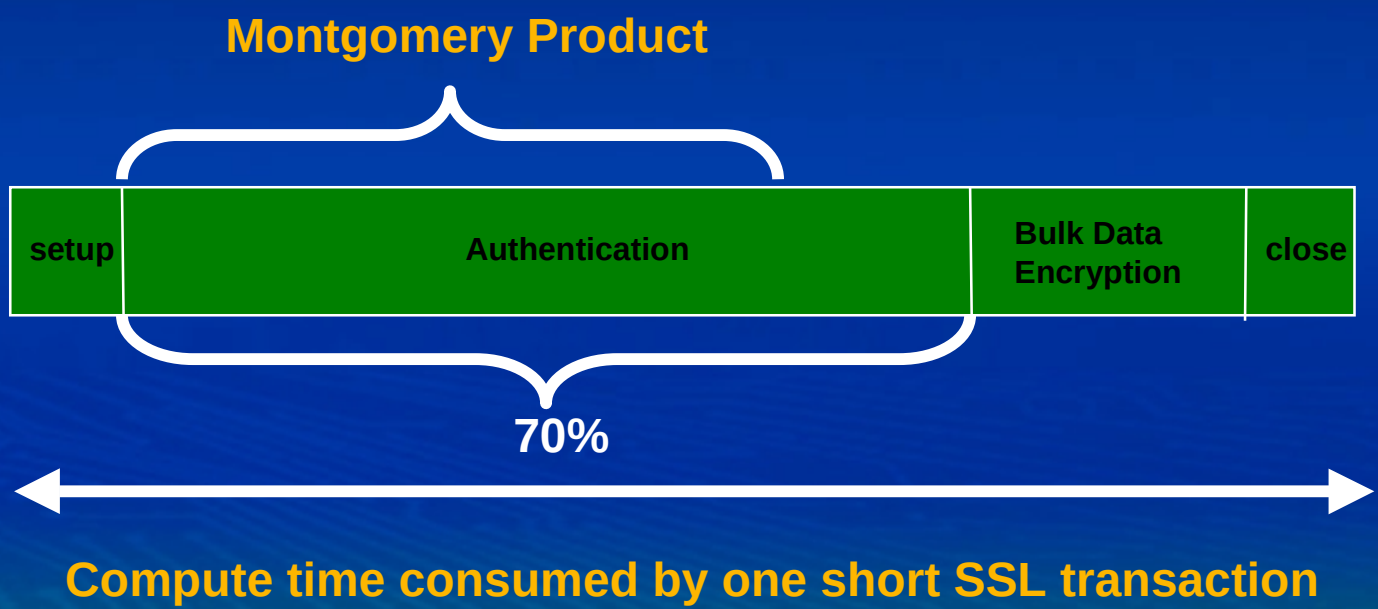


Source - Intel Lab research

RSA Core dominates small SSL transactions typical for e-Commerce

Computation in SSL

- Goal: Increase the number of secure transactions
  - Identify ***server*** performance issues in SSL
  - One server may deal with hundreds of clients



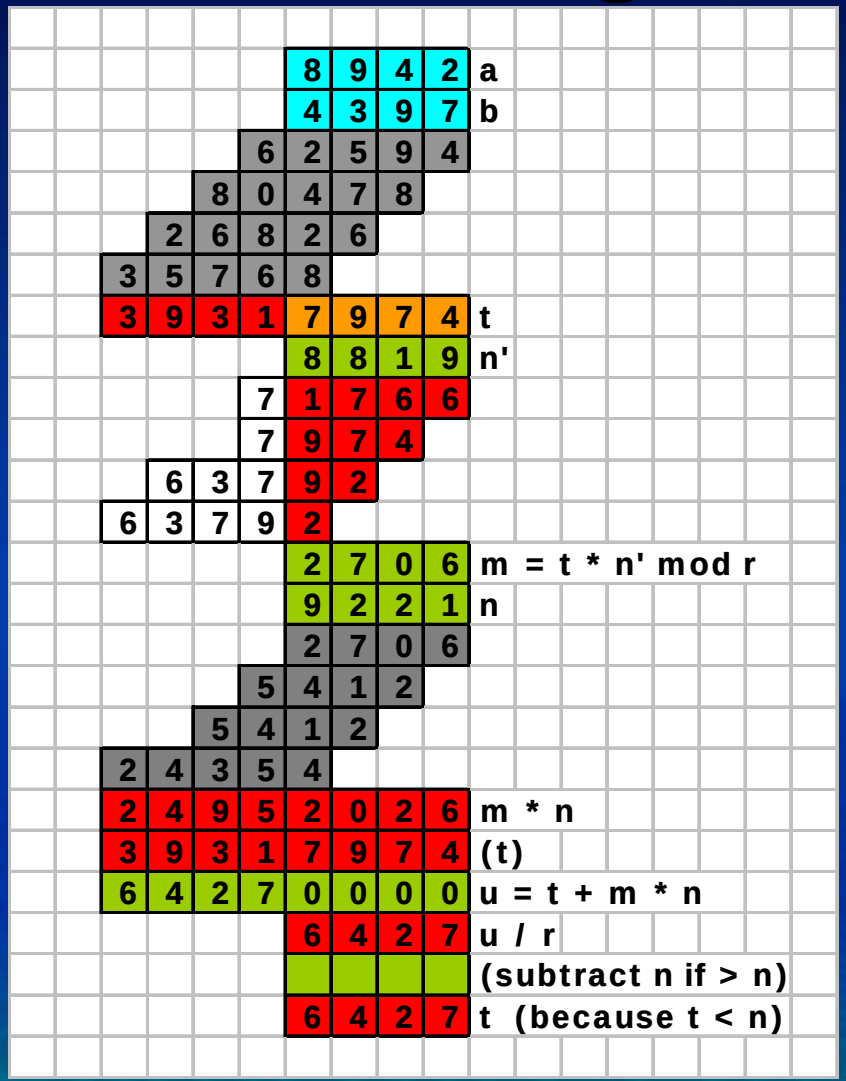
Source - Intel Lab research

The Bottleneck

$$M = E^d \bmod N$$

- Parameters are all big numbers
  - At least 512 bits
- The modulo operation is very expensive
  - Montgomery Product (MP) is standard implementation

The Montgomery Product



- MP avoids trial division
- One 1024-bit RSA decrypt requires around 1500 512-bit MPs
- Approximate number of CPU instructions per decrypt:
  - 32-bit architecture
    - 1.5 million multiplies
    - 1.5 million adds
  - IA-64 architecture
    - 375K multiplies
    - 375K adds

• One RSA decrypt

Impact of MP Optimization

- MP is the bottleneck in RSA algorithm
- Same calculation is a bottleneck in other public-key algorithms
  - DSA
  - Diffie-Hellman
  - El Gamal
- Payoff is big for all PKCS*

*Public Key Crypto System

Getting Breakthrough Performance from Itanium™ Processors

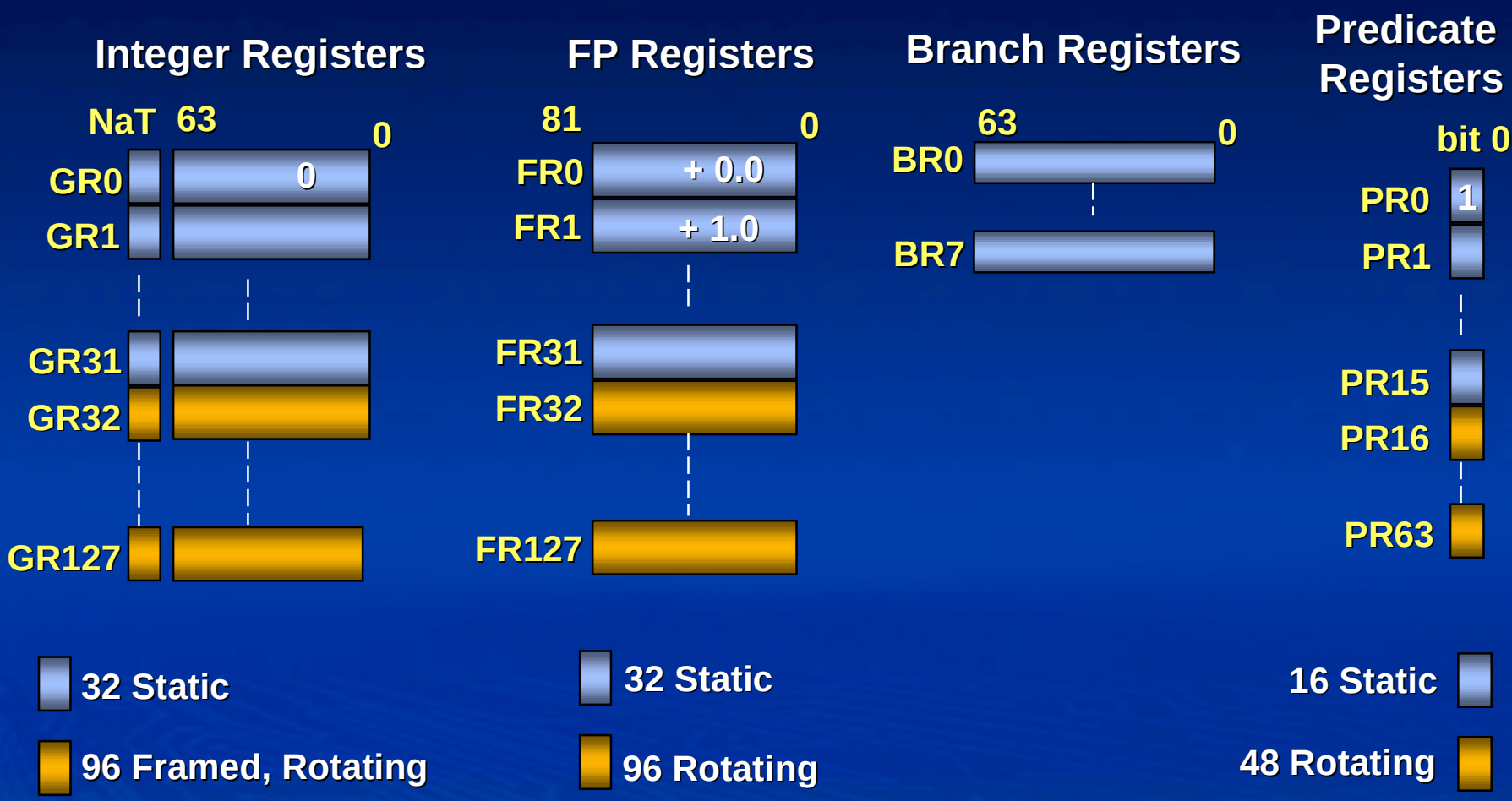
- Architectural Features
- Optimization Approaches
- Performance Results

Multiple Execution Units

- Two Integer units
- Two Floating Point units
- Two Integer/Memory units
  - Memory units can do integer instructions
  - Single cycle latency for most integer instructions

Can do several pieces of computation in parallel

Large Register Set



Maintain large computations in registers

Integer Multiply and Add

- 64-bit *xma* Integer Multiply & Add Instruction
- 64-bit upper or lower result
- Executes on either FP unit
- Fully pipelined

- More computation per instruction
- Well suited to large number multiply

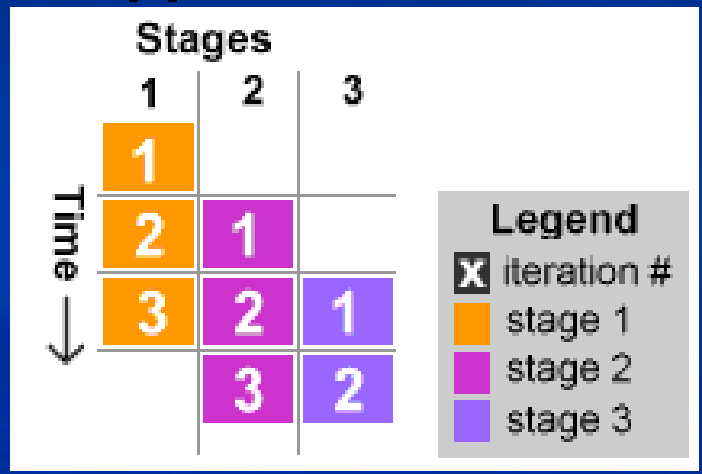
Parallel Instruction Execution

Unit	*Instruction*	*Function*
FPU 0	`xma.lu`	Integer Multiply with Add
FPU 1	`xma.hu`	Integer Multiply with Add
Integer 0	`add`	Add
Integer 1	`p = cmp.ltu`	Integer Compare for Carry
Memory 0	`getf.sig`	Move FP Reg to General Reg
Memory 1	`(p) add`	Predicated Add for Carry

- Six computations in one CPU cycle
- Predicates remove branches
- Two xma's give 128-bit result

Software Pipelining

- Software technique:
  - Different steps in loop execute in parallel
  - Prelude and postlude to start and stop pipeline
- Itanium™ Processor Hardware support
  - Register rotation
  - Special branch instructions
  - Predicated instructions
  - Supported by the compiler



- Reduce code size and complexity
- IA-64 + compilers do pipelining

Performance Considerations

- Match the algorithm to the architecture
- Coding considerations
  - C versus assembly
  - Loop versus unrolling
  - Software pipelined or not

Tuning variations

- Intuitive algorithm
  - Straight C
  - Slightly tuned C (128-bit product)
  - Software Pipelined C
- ‘Deconstructed’ algorithm
  - Software Pipelined Assembly
  - Unrolled Assembly

Let's Do the Math

- **Montgomery Product**
  - **1024-bit RSA decrypt (w/CRT*)**

    CRT = two 512-bit exponentiations

    each 512-bit exponentiation is about 750 MPs

    therefore 2 * 750 MPs = 1500 MPs

    - **Overhead:**
      - Some for managing exponentiation
  - **MP Uses about 2.5 512-bit multiplies**
    - **This equates to:**

      2.5 * 64 = 160 multiplications per MP
    - **Overhead is:**

      Another 50% other work

*CRT - Chinese Remainder Theorem

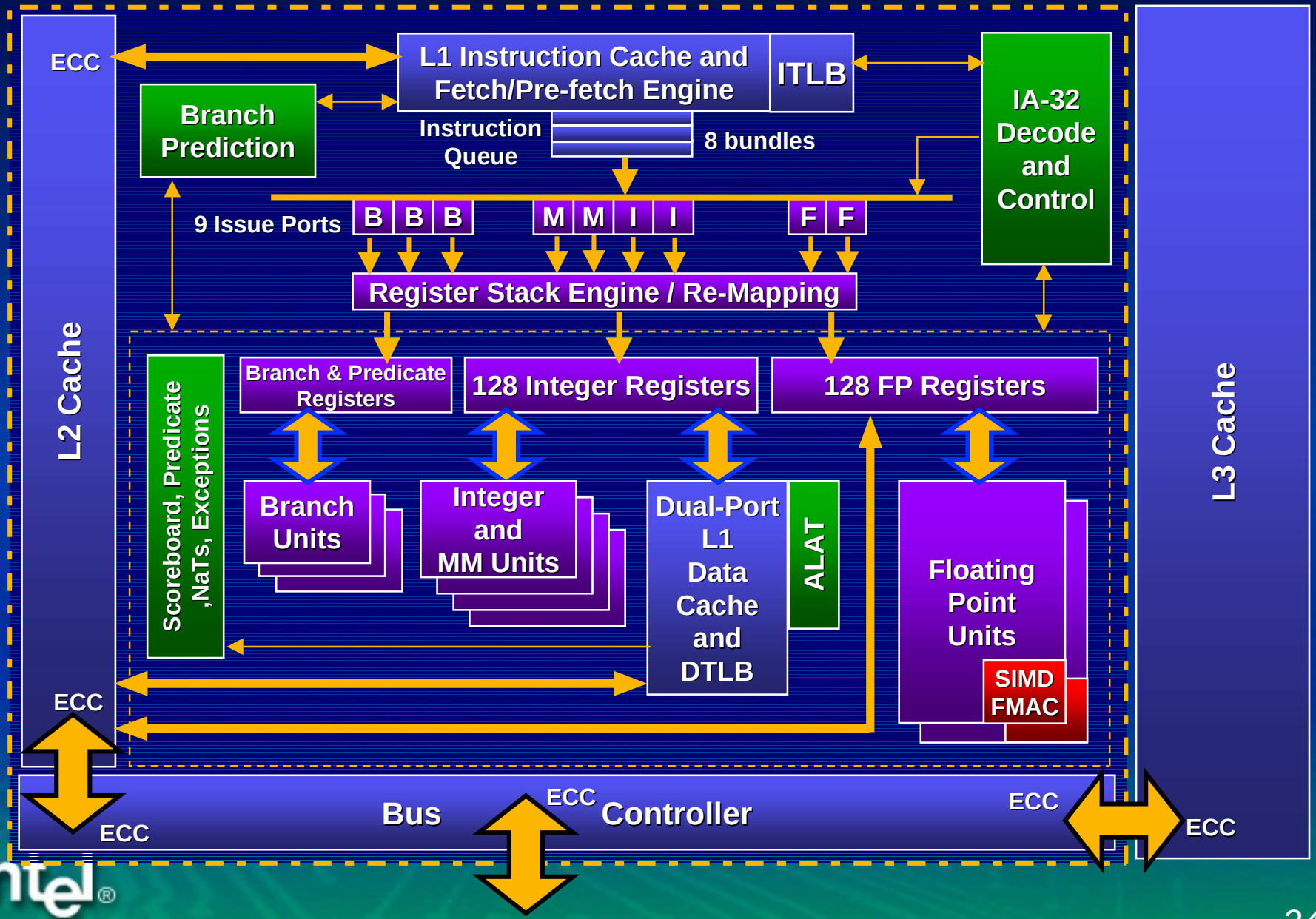
Summary

- Breakthrough performance on public key algorithms for Itanium™ processor
  - The right architecture
  - The right instruction set
- Tools for development are available now
  - http://developer.intel.com/vtune
- Additional Security Links
  - http://developer.intel.com/design/security/rsa2000

IA-64 delivers more secure transactions to more users

Backup

Intel® Itanium™ Processor Block Diagram



Itanium™ Code Example

```
for (i=0;i<n;i++)
  y[i]=x[i]+1
```

Pseudo Assembly Code

```
loop:
  (p16) ld1 r32   = [r12],1
  (p17) add r34   = 1,r33
  (p18) st1 [r13] = r35,1
```

Montgomery Product

Assume r, n relatively prime, then:

$\{ i \bullet r \bmod n \mid 0 \le i \le n - 1 \}$ is a complete residue system

Given two *n*-residues a', b' (where $x' = x\ r \bmod n$), then

$$MP = a' \bullet b' \bullet r^{-1} \bmod n$$
$$= a \bullet r \bullet b \bullet r \bullet r^{-1} \bmod n$$
$$= a \bullet b \bullet r \bmod n$$

Choose n' such that

$$r \bullet r^{-1} - n \bullet n' = 1$$

then this is MP algorithm

$t = a' \bullet b'$
$m = t \bullet n' \bmod r$
$u = (t + m / n)\ r$
if $u \ge n$ then $u = u - n$
$MP = u$

Exponentiation Heuristic

To raise *a* to exponent *x*:

Convert *x* to binary form

Let $r = a^0$

Scanning bits from MSB to LSB:

$r = r^2$

If bit is 1 then

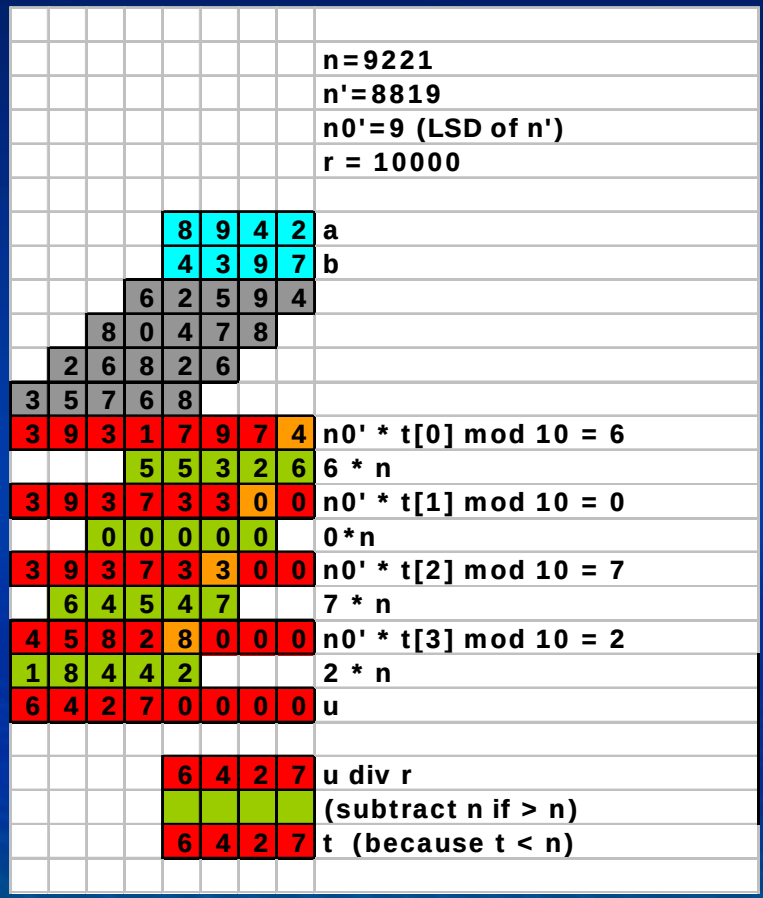
$r = r \bullet a$

Example:

$x = 55, x = 110111_b$

$1 \rightarrow a^0 \bullet a^0 \bullet a^1 = a^1$
$1 \rightarrow a^1 \bullet a^1 \bullet a^1 = a^3$
$0 \rightarrow a^3 \bullet a^3 = a^6$
$1 \rightarrow a^6 \bullet a^6 \bullet a^1 = a^{13}$
$1 \rightarrow a^{13} \bullet a^{13} \bullet a^1 = a^{27}$
$1 \rightarrow a^{27} \bullet a^{27} \bullet a^1 = a^{55}$

Computing MP (variation)



Computing MP (variation)

				8	9	4	2	a
				4	3	9	7	b
			6	2	5	9	4	
			6	2	5	9	4	4*n0'=6 mod 10
			5	5	3	2	6	6*n
		1	1	7	9	2	0	
		8	0	4	7	8		
		9	2	2	7	0	0	0*n0'=0 mod 10
		0	0	0	0	0		0*n
	0	9	2	2	7	0	0	
	2	6	8	2	6			
	3	6	0	5	3	0	0	3*n0'=7 mod 10
	6	4	5	4	7			7*n
1	0	1	5	0	0	0	0	
3	5	7	6	8				
4	5	9	1	8	0	0	0	8*n0'=2 mod 10
1	8	4	4	2				2*n
6	4	2	7	0	0	0	0	
				6	4	2	7	u div r
								(subtract n if > n)
				6	4	2	7	t (because t < n)

Intuitive Algorithm

```
// A[0..i-1] = Multiplicand 1
// B[0..j-1] = Multiplicand 2
// T[0..i+j-1] = Result
// word = relevant word size
// words = number of words in multiplicand

BigMultiply (A[], B[], T[])
{
        oneword i,j,carry;
        twoword result;
        for(carry = 0,i = 0; i < words; i++)
        {
                for(j = 0; j < words, j++)
                {
                        result = a[i] * b[j] + t[i+j] + carry;
                        t[i+j] = LowWord(result);
                        carry = HiWord(result);
                }
        }
        t[i+j] = carry;
}
```

For more information

http://developer.intel.com/design/security

- Intel Security Technologies
- Updated speaker presentation
- Other Intel Security Events